Modeling

Our project is mainly centered on how to improve the storage density and avoid the mistakes that may occur during the process of designing a stable, high-density DNA information storage system. There are two main technological processes, encoding and decoding, in the system.

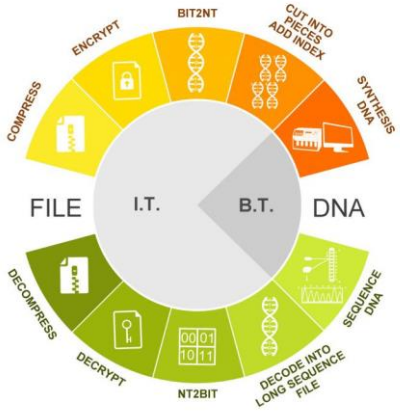

Figure 1: system flow diagram

Encoding

Compression: ZIP algorithm

We used bzip2 algorithm which renowned as a high-quality data compression algorithm to compress the file. It typically compresses files to within 10% to 15% of the best available techniques, whilst being around twice as fast at compression and six times faster at decompression. After this process, we get a bz2 compression file.

Encryption: ISAAC64 encryption algorithm

Next, we use ISAAC64 encryption algorithm to encrypt the bz2 file. After you input your own password, ISAAC generates a pseudorandom stream of bits (a keystream). As with any stream cipher, these can be used for encryption by combining it with the plaintext using bit-wise exclusive-or; decryption is performed the same way (since exclusive-or with given data is an involution). After this process, we can get a sufficiently random binary file.

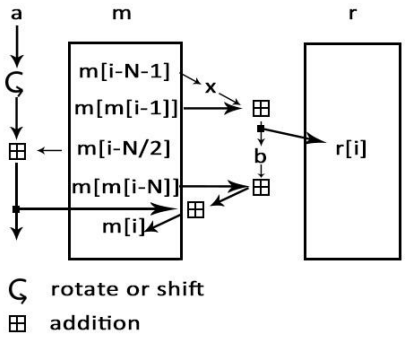

Figure 2: encryption process

Bit-to-nt conversion: quaternary system

In bit-to-nt conversion process, we use the idea of the quaternary system. As we all know, there are four basic groups—A, T, C, G, and it can be seen as a quaternary system. In bit-to-nt conversion, one byte of bits converts into four bytes of A, T, C, G, using the scheme illustrated in Table 1.

Bits	00	01	10	11
Base	A	C	G	T

Table 1: bit-to-nt conversion

We read out the binary string S_0 of the binary file generated in the last process. Use bit-to-nt conversion to convert S_0 into a DNA string S_1. We get a long DNA sequence.

Fragmenting & indexing

Write len( ) for the function that computes the length of a string, and define $n = \text{len}(S_1)$. Represent n in base-4 and prepend '0's to generate a string S_2 of quaternary such that $\text{len}(S_2) = 15$. Form the string concatenation

$$S_4 = S_1.S_3.S_2 \qquad (1)$$

(the symbol '.' means the connection of two srings)

where S_3 is a string of at most 49 '0's chosen so that $\text{len}(S_4)$ is an integer multiple of 50.

Convert S_2 and S_3 to DNA strings S_2' and S_3' using the scheme illustrated in Table 2.

Quaternary number	0	1	2	3
Base	A	C	G	T

Table 2: quaternary-to-nt conversion

Recode the DNA string $S_3' . S_2'$ from the second character to S_2'' with repeated nucleotides as few as possible using the scheme illustrated in Table 3.

previous nt written	next nt to recode			
	A	C	G	T
A	C	G	T	A
C	G	T	A	C
G	T	A	C	G
T	A	C	G	T

Table 3: recoding table

Form

$$S_5 = S_1.S_2'' \qquad (2)$$

Even-odd check

Define $N = \text{len}(S_5)$. Split S_5 into overlapping segments of length 200 nt, each

offset from the previous by 50 nt. This means there will be $\frac{N}{50} - 3$ segments, conveniently indexed

$$i = 0,1, \ldots, \frac{N}{50} - 4 \qquad (3)$$

segment i is denoted F_i and contains (DNA) characters

$$50i, \ldots, 50i + 199 \qquad (4)$$

of S_5. If i is odd, reverse complement F_i.

Let $i4$ be the base-4 representation of i, appending enough leading '0's so that

$$\text{len}(i4) = 12 \qquad (5)$$

Recode i4 using the same strategy in Table 2 & Table 3 above, and i4 is represented in nt.

Compute P as the $\text{sum}(\text{mod } 4)$ of the even-positioned quaternary in $i4$.

$$P = (i4_2 + i4_4 + i4_6 + i4_8 + i4_{10} + i4_{12}) \text{ mod } 4 \qquad (6)$$

P acts as a 'parity quaternary'—analogous to a parity bit—to check for errors in the encoded information about i.Form the indexing information string

$$IX = i4.P \qquad (7)$$

(comprising 12+1=13 nt)

Append the DNA-encoded version of IX to F_i to give indexed segment F_i'.

Then form F_i'' by prepending A or T and appending C or G to F_i' —choosing between A and T, and between C and G, randomly if possible but always such that there is no repeated nt. This ensures that we can distinguish a DNA segment that has been reverse complemented during DNA sequencing from one that has not—the former will start with G|C and end with T|A; the latter will start A|T and end C|G.

The segment F_i'' are synthesized as actual DNA oligonucleotides and stored, and may be supplied for sequencing.

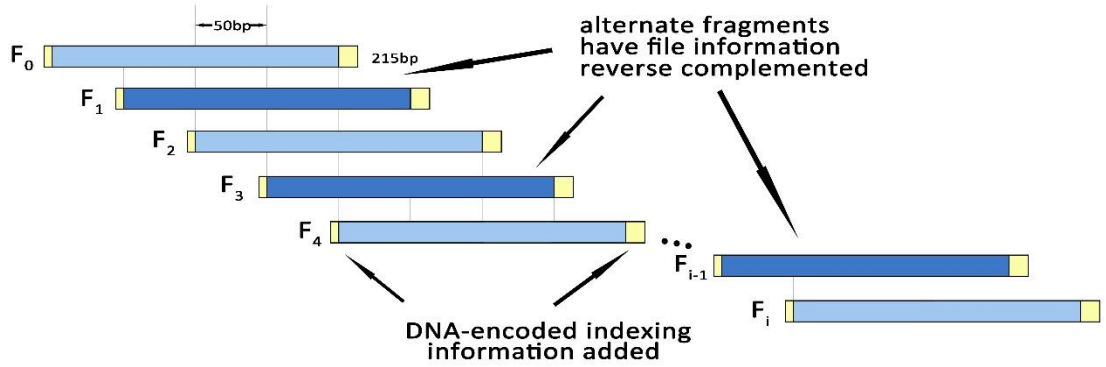

Figure 3: Fragmenting & Indexing

Safety testing: BLAST

In order to check whether the sequences we generated are safe, we use BLAST to compare the sequences with the Biobricks database. We use all the Biobricks sequences to establish a FASTA format file, and use BLAST to format it to set up a BLAST database. Then through local BLAST, we compare the sequences generated with the Biobricks sequences to confirm that the sequences are out of bio-function.

Decoding

Recognition of the front and the end of a sequence

Reverse complementation during the DNA sequencing procedure(e.g. during PCR reactions) can be identified for subsequent reversal by observing whether fragments start with A|T and end with C|G, or start with G|C and end with T|A.

Reading and check of index

With these two 'orientation' nt removed, the remaining 213 nt of each segment can be split into the first 200 'message' nt and the remaining 13 'indexing' nt. Decode the 'indexing' nt to quanternary using the reverse of the encoding tables in in Table 2 & Table 3 above, and use P to check the correctness of i4.

Correction of segments through four-fold redundancy

Use i to determine the location of each fragment. Split each fragment into segments of length 50 nt. As we provide a four-fold redundancy, we compare the 50-nt segments which are in the same location and inaccurate bases can be corrected by using majority vote. Connect all the segments to a whole DNA sequence.

Decoding, decryption and decompression

Decode the DNA sequence by using the reverse of the encoding table in Table 1 above, and use ISAAC64 to decrypt it, and then decompress. We get the original file.

Fuzzy matching with high-throughput sequencing

Errors introduced during DNA synthesis, storage or sequencing could lead to various nt insertion, deletion or substitution. Recovery of information from fragments with such errors may be possible via PCR amplification and high-throughput sequencing.

Example

In order to make our encoding process more understandable and clearer, we show an example here.

With a given text file, we read out its binary file, then compress it with bzip2 algorithm, and use a pseudorandom stream of bits to encrypt it. Then use bit-to-nt conversion to convert the binary numbers to bases. Shown in Figure 4.

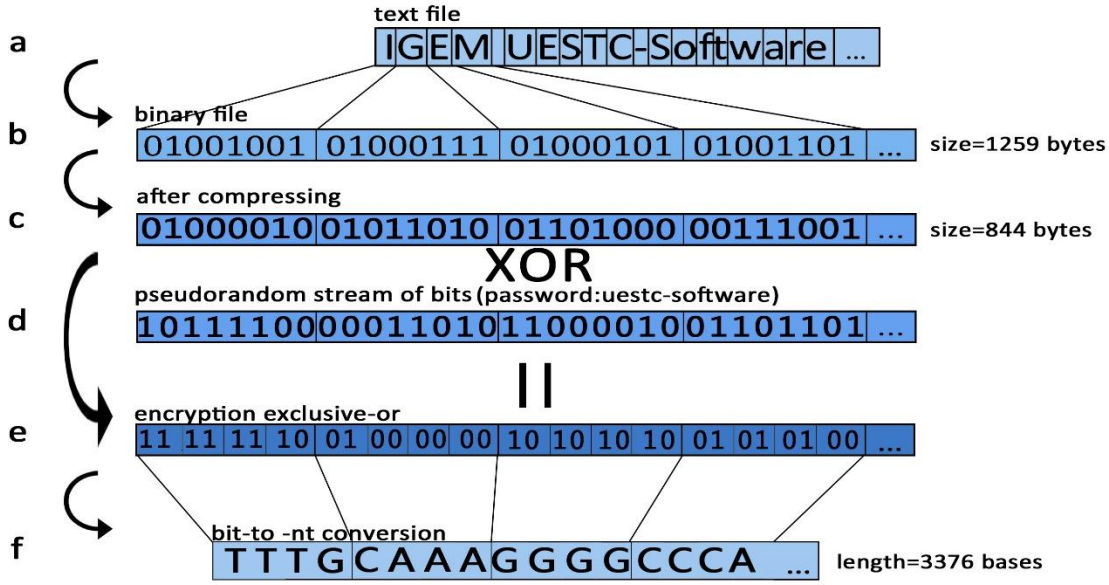

Figure 4: Schematic of DNA information storage system. The computer file (in any format, e.g. text) shown in (a) is read in binary format (b), and use bzip2 to compress it (c). Then through ISAAC64 encryption algorithm use pseudorandom stream of bits (d) and exclusive-or to generate a new binary string (e). Then one byte of bits converts into four bytes of A, T, C, G (e).

1. Towards the text file "IGEM UESTC-SOFTWARE", its binary coding is

	I	G	E	M	(space)	U	E	S
S =	01001001	01000111	01000101	01001101	00100000	01010101	01000101	01010011
	T	C	-	S	o	f	t	w
	01010100	01000011	00101101	01010011	01101111	01100110	01110100	01110111
	a	r	e	...				
	01100001	01110010	01100101	...				

(file size=1259 bytes)

2. After compressing, its compressed coding is:

S' = 01000010 01011010 01101000 00111001 00110001 01000001 01011001 00100110
01010011 01011000 11110011 ..... (compressed file size=844 bytes)

3. With the pseudorandom stream of bits that ISAAC64 generated (password: uestc-software):

(lack of data, data need to be caculate)

Using XOR operation:

S' XOR (pseudorandom stream of bits)

We get a new binary string S_0:

S_0 = 11111110 01000000 10101010 01010100 10100010 11100110 01001011 11001011
11011011 00000001 00001110 ..... (encrypted file size=844 bytes)

4. Using the scheme illustrated in Table 1, we convert the bytes of S_0 into bases—A, T, C, G.

S_1 = TTTG CAAA GGGG CCCA GGAG TGCG CAGT TAGT
TCGT AAAC AATG ..... (length=3376 nt)

5. $n = \text{len}(S_1) = 3376$, which is 310300 in base-4. So:

S_2 = 000000000 310300 (length 15)

S_3 = 000000000 (length 9)

$\text{len}(S_4) = \text{len}(S_1) + \text{len}(S_2) + \text{len}(S_3) = 3376 + 15 + 9 = 3400 = 50 * 68$

6. Using the scheme illustrated in Table 2, convert S_2 and S_3 to DNA:

$S_3'.S_2'$ = AAAAAAAAAAAAAAAAAATCATAA

Recode the DNA string $S_3'.S_2'$ from the second character to S_2'' with repeated nucleotides as few as possible, using the scheme illustrated in Table 3.

S_2'' = ACGTACGTAACGTACGTAAGTTAC

$S_5 = S_1.S_2''$ = TTTG CAAA GGGG CCCA GGAG TGCG CAGT TAGT
TCGT AAAC AATG .... ACGT ACGT
AACG TACG TAAG TTAC (length 3400)

7. $N = \text{len}(S_5) = 3400$. We split S_5 into overlapping segments of length 200 nt, each offset from the previous by 50 nt.

S_5 will be split into overlapping segments F_i of length 200 nt for

$$i = 0 \ldots \frac{3400}{50} - 4,\ i = 0,1, \ldots, 64$$

With overlapping parts underlined for illustration, F_0 to F_i are:

F_0 = TTTGCAAAGGGGCCCAGGAGTGCGCAGTTAGTTCGTAAACAATGTGACAA GAAGTAAAGATCACCCCATCCGTACGTTGGAAACGTGACTATTTAGGAGC
TCTAAGCCCAATGGCTACTCATCCCAGGCCCACAACGTTGCGTAAGGGCC TAAACCGTCGTGAGAGGCGAGGAGACCGGTTTGCGAGCAACGGACTGCAG
F_1 = GAAGTAAAGATCACCCCATCCGTACGTTGGAAACGTGACTATTTAGGAGC TCTAAGCCCAATGGCTACTCATCCCAGGCCCACAACGTTGCGTAAGGGCC
TAAACCGTCGTGAGAGGCGAGGAGACCGGTTTGCGAGCAACGGACTGCAG GGAGATGATCATAGTCGCATTCGCCCCATACTAGAACATTTTGTATGGCT
.....
.....

8. Only $i = 1,3, \ldots, 63$ are odd, so F_i is reverse complemented:

F_1 = AGCCATACAAAATGTTCTAGTATGGGGCGAATGCGACTATGATCATCTCC CTGCAGTCCGTTGCTCGCAAACCGGTCTCCTCGCCTCTCACGACGGTTTA
GGCCCTTACGCAACGTTGTGGGCCTGGGATGAGTAGCCATTGGGCTTAGA GCTCCTAAATAGTCACGTTTCCAACGTACGGATGGGGTGATCTTTACTTC
F_3 = ...
.....

9. For $i = 0, i4 = 000000000000$ (length 12) and the sum (mod 4) of the even-positioned quaternaries of $i4$ is

$$P = (0 + 0 + 0 + 0 + 0 + 0)(\text{mod } 4) = 0$$

For $i = 1, i4 = 000000000001$

$$P = (0 + 0 + 0 + 0 + 0 + 1)(\text{mod } 4) = 1$$

10. For $i = 0, IX = i4. P = 0000000000000$

For $i = 0, IX = i4. P = 0000000000011$.

So:

F_0 = TTTGCAAAGGGGCCCAGGAGTGCGCAGTTAGTTCGTAAACAATGTGACAA GAAGTAAAGATCACCCCATCCGTACGTTGGAAACGTGACTATTTAGGAGC
TCTAAGCCCAATGGCTACTCATCCCAGGCCCACAACGTTGCGTAAGGGCC TAAACCGTCGTGAGAGGCGAGGAGACCGGTTTGCGAGCAACGGACTGCAG
ACGTACGTACGT A
F_1 = AGCCATACAAAATGTTCTAGTATGGGGCGAATGCGACTATGATCATCTCC CTGCAGTCCGTTGCTCGCAAACCGGTCTCCTCGCCTCTCACGACGGTTTA
GGCCCTTACGCAACGTTGTGGGCCTGGGATGAGTAGCCATTGGGCTTAGA GCTCCTAAATAGTCACGTTTCCAACGTACGGATGGGGTGATCTTTACTTC
ACGTACGTACGA C

.....

11. Prepend A|T and append C|G (in this example we have three random choice, at the front of F_0'', the end of F_0'', and the end of F_1''):

F_0 = A TTTGCAAAGGGGCCCAGGAGTGCGCAGTTAGTTCGTAAACAATGTGACAA GAAGTAAAGATCACCCCATCCGTACGTTGGAAACGTGACTATTTAGGAGC
TCTAAGCCCAATGGCTACTCATCCCAGGCCCACAACGTTGCGTAAGGGCC TAAACCGTCGTGAGAGGCGAGGAGACCGGTTTGCGAGCAACGGACTGCAG
ACGTACGTACGT A G (length=1+200+12+1+1=215 bp)
F_1 = T AGCCATACAAAATGTTCTAGTATGGGGCGAATGCGACTATGATCATCTCC CTGCAGTCCGTTGCTCGCAAACCGGTCTCCTCGCCTCTCACGACGGTTTA
GGCCCTTACGCAACGTTGTGGGCCTGGGATGAGTAGCCATTGGGCTTAGA GCTCCTAAATAGTCACGTTTCCAACGTACGGATGGGGTGATCTTTACTTC
ACGTACGTACGA C G (length=1+200+12+1+1=215 bp)

.....